Interoperating Context Discovery Mechanisms*

Tom Broens[1], Remco Poortinga[2] and Jasper Aarts[1]

[1]Centre for Telematics and Information Technology, ASNA group, University of Twente
P.O. Box 217, 7500 AE Enschede, The Netherlands

[2]Telematica Instituut, PO Box 589, 7500 AN Enschede, The Netherlands

Abstract. Context-Aware applications adapt their behaviour to the current situation of the user. This information, for instance user location and user availability, is called context information. Context is delivered by distributed context sources that need to be discovered before they can be used to retrieve context. Currently, multiple context discovery mechanisms exist, exhibiting heterogeneous capabilities (e.g. communication mechanisms, and data formats), which can be available to context-aware applications at arbitrary moments during the application's lifespan. In this paper, we discuss a middleware mechanism that enables a (mobile) context-aware application to interoperate transparently with different context discovery mechanisms available at run-time. The goal of the proposed mechanism is to hide the heterogeneity and availability of context discovery mechanisms for context-aware applications, thereby facilitating their development.

1 Introduction

The Service-Oriented Architecture (SOA) paradigm provides a promising approach to develop distributed applications. In this paper, we are concerned with (distributed) context-aware applications, which are applications that use the current situation, called context, to adapt their behaviour [1]. There are numerous examples of possible types of context, depending on the goal of the application. Examples of context are user location and availability, room temperature, and available bandwidth on a communication link. Context-aware applications use information on the context to adapt their functionality with the aim of improving the quality of the service offered to the user. For example, a context-aware 'buddy navigation application' that can offer quick and personalized navigation to available buddies, based on the location of the user and his buddies, correlated with the availability of the buddies. Context is usually provided by various distributed context sources (e.g. GPS sensors for location information, calendar for scheduling or availability information, MSN messenger for buddy information, weather stations for current weather conditions). The application

* This work was partly supported by the Freeband AWARENESS Project. Freeband is sponsored by the Dutch government under contract BSIK 03025. Additionally, this work was partly supported by the European Commission as part of the IST-IP AMIGO project under contract IST–004182.

Broens T., Poortinga R. and Aarts J. (2007).
Interoperating Context Discovery Mechanisms.
In *Proceedings of the 1st International Workshop on Architectures, Concepts and Technologies for Service Oriented Computing*, pages 75-84
DOI: 10.5220/0001348400750084
Copyright © SciTePress

environment may contain several useful context sources at any point in time, however, due to for example the mobility of the application or the context sources, the number and identity of context sources may change over time. This requires mechanisms to discover context sources before an application can retrieve context information. Currently there is a trend towards middleware mechanisms that facilitate the development of context-aware applications [2]. Major contributions in this area are context management systems that facilitate the context exchange process, including, amongst others, the discovery of context sources. Consequently a vast amount of context discovery mechanisms exist, which have different capabilities and scope [2-5]. We believe it is unlikely that there will be one future commonly adopted context discovery mechanism. As implied by the diversity of currently available context discovery mechanisms, different application environments (e.g. ad-hoc environments, telco environments) require different mechanisms to exchange their context information. Therefore, the mechanisms, which context-aware applications have to use to discover context sources from these environments, will be diverse. Consequently, (mobile) context-aware applications are likely to be exposed to multiple and changing context discovery mechanisms during their lifespan. Without supporting mechanisms to cope with this aspect, developers have to design and incorporate interoperability mechanisms or consider every possible mechanism statically in their application. Besides the required, substantial, programming effort, this also distracts from the primary task of developing context-aware applications.

In our view, there are three approaches to interoperate context discovery mechanisms:

Standardisation: every environment that wants to offer context discovery uses one standard context discovery mechanism. However, as already indicated, due to the heterogeneity and different requirements of the application environments, this is not feasible or likely.

Bridging: every environment has a different discovery mechanism that is internally bridged to other discovery mechanisms by bridging components or code.

Homogenising: every environment has different discovery mechanisms that are homogenized by a generic middleware layer, optionally co-located with the application (see Figure 1 for a comparison of the bridging and homogenising approach).

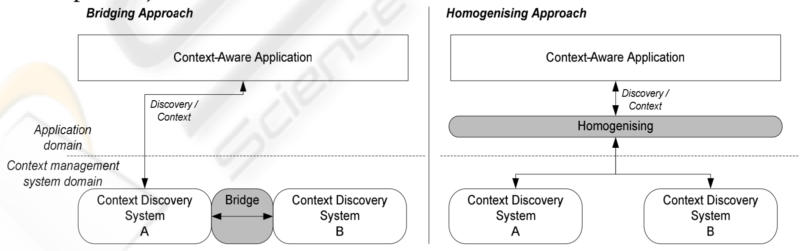
Bridging Approach
Context-Aware Application
Discovery / Context
Application domain
Context management system domain
Context Discovery System A
Bridge
Context Discovery System B
Homogenising Approach
Context-Aware Application
Discovery / Context
Homogenising
Context Discovery System A
Context Discovery System B

Fig. 1. Comparison of the bridging and homogenising approach.

In this paper, we explore the homogenising approach and propose a middleware mechanism that enables a context-aware application to interoperate transparently with different context discovery mechanisms that are available at run-time. The goal of the proposed mechanism is to hide the heterogeneity and availability of context discovery mechanisms for context-aware applications, thereby facilitating their development. We envision the proposed homogenising approach as part of a comprehensive SOA infrastructure to support composable context-aware services. The AWARENESS project (http://awareness.freeband.nl) also explores the bridging approach which is discussed in [3, 6]. The remainder of this paper is structured as follows: section 2 discusses a motivating scenario of a context-aware application that uses multiple context discovery mechanisms. Section 3 presents an analysis of current context discovery mechanisms. This analysis is used to derive requirements for our interoperability mechanism. Section 4 discusses our contribution towards interoperability of context discovery mechanisms. Section 5 presents our proof-of-concept prototype and gives an initial qualitative evaluation. Finally, in section 6, we will end with some conclusions.

2 Scenario

In this section, we reconsider the context-aware 'buddy navigation application' and use it to identify key difficulties that application developers face when developing context-aware applications.

Dennis is a young adult, always wanting to be in contact with his friends. He has a mobile device running the 'buddy navigation application'. This application is able to navigate to available buddies by using location and availability context information of him and his friends. Dennis notices that Monica is in the mall and available for a cup of coffee. He decides to visit her. He instructs the 'buddy navigation application' to help him find her. Inside Dennis' home, a RFID based location context source, found by his home context discovery mechanism, provides accurate location of Dennis. From Monica no precise location source is available in Denis's home, it is only known that she is somewhere in the mall. The 'buddy navigation application' instructs Dennis to take the car to the mall. When Dennis leaves his home, to go on his way to Monica, his home discovery mechanism becomes unavailable. The application switches to a cell based location context source found by the context discovery mechanism of his telecommunication provider. On entering the mall Monica is in, accurate context information on Monica's location becomes available, offered by a Bluetooth beacon context source found by the context discovery mechanisms of the mall. The buddy navigation application pops up a map of the mall, to instruct Dennis how to walk to the book store where Monica is currently shopping.

From the scenario, we can identify the following difficulties, related to context discovery that application developers face when developing context-aware applications:

- Finding of context sources through different context discovery mechanisms (e.g. 'home', 'telecommunication', and 'mall' context discovery mechanisms).

Fluctuating availability of context discovery mechanisms (e.g. when Dennis leaves his home his home context discovery mechanism is not available any more).

In this paper, we propose a middleware mechanism that has the goal to support context-aware applications to interoperate with multiple heterogeneous context discovery mechanisms, considering their availability during the lifetime of the application.

3 Analysis of Current Context Discovery Mechanisms

As indicated, many context discovery mechanisms exist. We analysed a subset of these mechanisms consisting of four approaches developed in the AWARENESS project (CMF, CCS, CDF, JEXCI)[3], and one approach developed in the AMIGO project (CMS)[7] and four external approaches (Context Toolkit [5], PACE [2], Solar [8], and JCAF[4]). The result of our analysis is presented in Table 1. The analysis consisted of reviewing the following aspects of the different discovery mechanisms:

Interaction mechanism: what interaction mechanisms do the analyzed discovery mechanisms support?
Interaction syntax: what information is expressed in the context discovery request?
Interaction format: what is the data format of the request and response?

Table 1. Context discovery mechanisms analysis results.

	Interaction mechansism		Interaction syntax			Interaction format
Frameworks	Req-Resp	Sub-Not	Entity	Type	QoC	
CMF	v	v	v	v	v	RDF
CCS	v	v	v	v	v	SQL/PIDF
CDF	v	v	v	v	v	RDF/PIDF
Jexci	v	v	v	v	v	*Negotiable (PIDF/java objects)*
CMS	v	v	v	v	v	RDF
Context Toolkit	v	v	v	v	-	XML
Pace	v	v	v	v	v	Context Modelling Language
Solar	v	v	v	v	-	?
JCAF	v	v	v	v	-	Java objects

We distinguished the following common aspects in the analysed approaches, which pose requirements on the capabilities our interoperability mechanism should offer to context-aware applications it supports:

The ‘request-response’ and ‘subscribe-notify’ interaction mechanisms are offered.
Requests for context minimally specify an entity and context type (e.g. ‘Location’ of ‘Tom’).
Requests for context may contain Quality of Context [9, 10] criteria.

We distinguished the following uncommon aspects in the analysed approaches, which pose requirements on the heterogeneity our interoperability mechanisms should have to overcome:

Data models (syntax and semantics) of the request and response (ontology, simple strings, binary).
Communication technologies (e.g. web services, jini).

Intelligence inside the context discovery mechanism (e.g. reasoning, selection).

4 Context Discovery Interoperability Mechanism

The scenario in section 2 and the analysis in section 3 indicate the type of difficulties a context-aware application may typically encounter. A viable context discovery interoperability mechanism will have to hide these difficulties or, at least, diminish the burden placed on the context aware application and its developer to overcome these difficulties.

Summarizing, the problems such an interoperability mechanism has to solve can roughly be divided into the following categories:

(Un)availability of context discovery mechanisms.
Heterogeneous interaction behaviour and communication mechanisms.
Heterogeneous data syntax and semantics.

In this paper, we mainly focus on the first two aspects and syntactic interoperability. We acknowledge the importance of having both syntactic and semantic interoperability; however, this is out of the scope of this paper and we consider this future work.

The pivotal requirement is the ability for a context discovery interoperability mechanism to dynamically adapt to different context discovery mechanisms appearing and disappearing. Based on the knowledge of which context discovery mechanisms are currently available, the interoperability mechanism should then change the way it discovers and retrieves context information on behalf of the applications it supports. By concentrating the specific functionality of the specific discovery mechanism in individual components that can be loaded and unloaded dynamically, the interoperability mechanism does not need to support all separate discovery mechanism simultaneously, and at the same time it is able to abstract from the specifics of individual discovery mechanisms, since this is hidden in the components itself. We call these environment specific components *context discovery adapters* (see Figure 2). For storing and retrieving these adapters at run-time, an *adapter supplier* is defined, which is a repository for discovery adapters. By allowing multiple adapter suppliers to co-exist (e.g. multiple environments), and not prescribing where these suppliers should be located (i.e. not restrict a repository to be co-located on the same host running the context-aware application), specific context discovery environments may provide their own adapter supplier, without losing the ability to use preferred adapters present in the co-located repository. The remaining item to be addressed is the monitoring of the availability of known context discovery mechanisms. Analogous to environment specific adapters, environment specific *monitors* are defined, which are responsible for detecting whether a particular context discovery mechanism is currently (still) available. Adapter suppliers present in a context discovery environment also provide these monitors; next to the discovery adapters mentioned earlier. The adapter supplier co-located on the same host as the context aware application also provides monitors for a set of predefined context discovery mechanisms.

An adapter supplier thus has the following responsibilities:

Provide adapters for the specific context discovery mechanism within its environment.

Provide monitors for the same specific context discovery mechanism, which allow the context discovery interoperability mechanism to detect its availability.

The latter responsibility of the adapter supplier also implies that for the first detection of context discovery mechanisms that can be used with the interoperability mechanism, it is sufficient to detect the presence of an adapter supplier. In order to leverage this approach, rather than creating a new discovery problem, the method to discover such a supplier should be standardised.Next to the different components, additional logic is necessary for the orchestration of the different adapters, monitors, and suppliers. This logic is provided by the *Discovery Coordinator*.

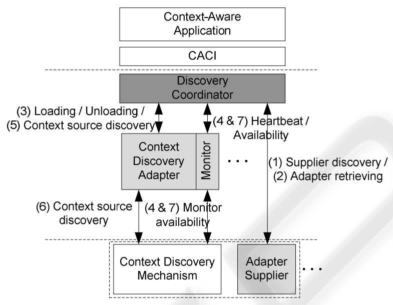
Context-Aware Application
CACI
Discovery Coordinator
(3) Loading / Unloading / (5) Context source discovery
(4 & 7) Heartbeat / Availability
Context Discovery Adapter
Monitor
. . .
(1) Supplier discovery / (2) Adapter retrieving
(6) Context source discovery
(4 & 7) Monitor availability
Context Discovery Mechanism
Adapter Supplier
. . .

Fig. 2. High-level design of the proposed Context Discovery Interoperability mechanism.

The proposed discovery interoperability mechanism is part of a more comprehensive effort towards a context binding transparency [11]. This transparency hides the complexity from the application developer of discovering, selecting, and binding to context sources, which he requires for his context-aware application. Furthermore, it maintains the binding with a context source, thereby coping with their dynamic availability. All these responsibilities are shifted to a middleware layer, coined CACI, which is co-located with the application. For more information on CACI see [12, 13]. Therefore, we integrated the presented interoperability mechanism with the CACI middleware. A typical scenario of the use of the discovery interoperability mechanism is as follows (see Figure 2 and 3): on start-up of the application and CACI, the Discovery Coordinator initiates a discovery of available adapter suppliers (1); this is done continuously e.g. by passive service discovery. When a supplier is found its available adapter/monitor combinations are downloaded (2). The monitor is started (3) to check the availability of the discovery mechanism (4). If it is indeed available, then the corresponding adapter is registered to the Discovery Coordinator, and passed on to CACI, which in turn will use the adapter to discover context sources (5 & 6). The monitor is continuously keeping track of the availability of the discovery mechanism it supports (7). If discovery mechanisms become unavailable, their adapters are deregistered with the coordinator, and indirectly with CACI. Although the figures suggest that only one monitor and adapter is active, multiple monitors and adapters can co-exist at the same time and can become active or inactive during the lifespan of the application.

5 Implementation

Summarising, the architecture contains the following components with their respective responsibilities:

> Context Discovery Adapter: component that translates between a specific context discovery framework and a context-aware application in the form of the CACI layer.
> Monitor: component that keeps track of the availability of a specific context discovery mechanism.
> Adapter Supplier: component that provides the Context Discovery Adapter and the Monitor to the Discovery Coordinator.
> Discovery Coordinator: component that orchestrates the interactions between the different components.

We made a proof-of-concept implementation of the discovery interoperability mechanism using an implementation of the OSGi component framework specification. OSGi implementations offer ‘a service-oriented, component-based environment for developers and offers standardized ways to manage the software lifecycle’ (see http://www.osgi.org). The open source OSGi implementation ‘Oscar’ was used as the basic implementation platform (http://oscar.objectweb.org). However, the prototype is also tested on the Knopflerfish OSGi (http://www.knopflerfish.org) implementation. For communication and discovery mechanisms the middleware of the IST Amigo project (http://www.amigo-project.org) was used, which amongst other things, provides components for easy Web Service communication (for both server and client side) and Web Services Dynamic Discovery (WS-Discovery), which uses multicast to discover web services of specific type and scope in the network. More specifically, the WS-Discovery mechanism, available from the Amigo project, was used as the ‘standard’ discovery mechanism for finding adapter suppliers. Every component in the architecture was implemented as a separate OSGi bundle (the OSGi name for a component), which has the added benefit that the bundle id can be used for identifying component instances. The OSGi framework is service-oriented and also deploys the concept of service listeners, which means that components can register themselves as being interested in services of a specific type. If a component that offers a specific type of service is installed and activated, all interested service listeners will be informed of that event. In the prototype implementation the service listener pattern is used by CACI to get notified whenever new Context Discovery Adapters become active after being downloaded by the discovery coordinator. A sequence diagram (see Figure 3) will help to derive the detailed functions of the different components and their respective interfaces. In the text below, italic text in brackets will indicate the interface name that is relevant for the mentioned interaction. In order to be discoverable by a Discovery Coordinator, an Adapter Supplier registers itself with a scope of ‘urn:CADC’ and a service type of ‘IAdapterSupplier’. After an adapter supplier is discovered, the Discovery Coordinator needs to retrieve the list of components provided by the adapter supplier (*IAdapterSupplier*), typically consisting of one Monitor and one or more Context Discovery Adapters. The Discovery Coordinator will download (using OSGi’s component downloading capabilities via http or file system) the components returned by the Adapter Supplier and start the Monitor by activating the Monitor component. If the Monitor successfully detects the context discovery mechanism supported by the adapters, it will start the adapter(s) and indicate the availability of the context discovery mechanism to the Discovery Coordinator (*IDiscoveryCoordinator*). The CACI framework will be notified of this since the started adapters provide a specific service, for which CACI has registered as a service listener. CACI will call the Context Discovery Adapters for performing Context Source Discovery (*IDicoveryAdapter*). The Monitor will keep checking the availability of its Context Discovery Mechanism. If it becomes unavailable, the Monitor will inform the Discovery Coordinator (*IDiscoveryCoordinator*) by deregistering and stopping the relevant adapters. Since stopping the adapters automatically means that the OSGi service they are offering disappears, CACI (as a service listener) will be notified of this event by the OSGi framework. Next to the Monitor, the Discovery Coordinator will continuously check for the availability of the Adapter Supplier via a straightforward heartbeat mechanism; essentially a dummy method call on the Adapter Supplier (*IAdapterSupplier*). If the supplier becomes unavailable, the coordinator will respond by stopping the Monitor belonging to the supplier that disappeared (*IMonitor*).

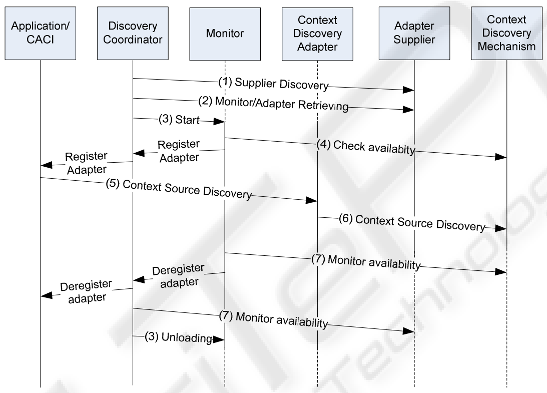

Fig. 3. Time-sequence of a scenario of loading and unloading of a context discovery adapter.

The following code snippets give interface definitions in pseudo-code of the different described components. These where already referred to in the text above.

```
IAdapterSupplier
String heartBeat(void)
Adapter[] listAdapters(void)
URL getAdapterReference(adapterID)

IDiscoveryAdapter
String getFriendlyName(void)
[] discoverContextProducers(request)

IMonitor
String getFriendlyName(void)
Long getComponentID(void)
IDiscoveryCoordinator
newMonitor(IMonitor)
monitorGone(IMonitor)
frameworkAvailable(IMonitor)
frameworkGone(IMonitor)
```

Implementations of the Adapter and Monitor components were made for the CCS, CMS, [3], and SimuContext [14] context management frameworks. They are currently being evaluated. For supporting other discovery mechanisms than the ones already implemented for the prototype, new Monitors and Adapters have to be developed. Since a large part of the functionality of the Monitor is equal for every type of context discovery mechanism, a new one can be implemented by deriving from the reference monitor component and implementing the template parts for the specific needs of the targeted context discovery mechanism. The specific Context Discovery Adapters are less generic than the Monitor, and should at least implement the IDiscoveryAdapter interface. The Discovery Coordinator and Adapter Supplier are generic and do not have to be (re-) implemented for new context discovery mechanisms, although the Adapter Supplier will have to be configured with the appropriate information for the context discovery mechanism it has to support (i.e. URLs of monitor and adapters).

6 Conclusions

This paper discusses work in progress towards a context discovery and delivery interoperability mechanism. In this paper, we focus mainly on the interoperability of context discovery mechanisms. By using the context discovery interoperability mechanism, application developers are relieved from programming mechanisms in their application to interoperate with different context discovery mechanisms that can appear and disappear at arbitrary moments during the life-span of the application. The mechanism actively searches for context discovery mechanisms and when found adds them to the scope of the interoperability mechanism by downloading discovery adapters made available by the discovery mechanisms. Furthermore, it continuously monitors the availability of discovery mechanisms and if one disappears, it removes the adapter from the interoperability mechanism. However, we acknowledge some aspects that are not addressed in this paper and which we consider possible future research:

- Interoperable context delivery: this paper focuses on the first step of the context discovery and delivery process, namely context (source) discovery. After context discovery, the actual context has to be transferred from the context source to the application, which poses a similar interoperability issue. The chosen dynamic adapter-based approach is designed for both discovery and delivery of context. However, in this paper the approach is only detailed for interoperating context discovery mechanisms. Therefore, we plan to further extend this mechanism with context delivery interoperability.
- Security: downloading ‘unknown’ code is considered a security risk. However, mechanisms exist to overcome this issue, such as code signing and firewalling [15].
- Semantic interoperability: in this paper, we focus on functional interoperability. However, interoperating the different data models used by the context discovery mechanisms is similarly important. Mechanisms exist to get semantic

interoperability, which could be used to extend the current mechanism (e.g. ontologies [16]).

References

1. Dey, A., Providing Architectural Support for Context-Aware applications. 2000, Georgia Institute of Technology.
2. Henricksen, K., et al., Middleware for Distributed Context-Aware Systems, in DOA 2005. 2005, Springer Verlag: Agia Napa, Cyprus.
3. Benz, H., et al., Context Discovery and Exchange, in Freeband AWARENESS Dn2.1, P. Pawar and J. Brok, Editors. Freeband AWARENESS Dn2.1, 2006.
4. Bardram, J., The Java Context Awareness Framework (JCAF) - A Service Infrastructure and Programming Framework for Context-Aware Applications, in Pervasive Computing. 2005: Munchen, Germany.
5. Dey, A., The Context Toolkit: Aiding the Development of Context-Aware Applications, in Workshop on Software Engineering for Wearable and Pervasive Computing. 2000: Limerick, Ireland.
6. Hesselman, C., et al. Interoperating Context Managment Systems for Pervasive Computing Environments. in forthcomming. 2007.
7. Ramparany, F., et al. An Open Context Management Information Management Infrastructure. in Intelligent Environments (IE'07). 2007. Ulm, Germany.
8. Chen, G. and D. Kotz. Solar: An open platform for context-aware mobile applications. in International Conference on Pervasive Computing. 2002. Zurich, Zwitserland.
9. Buchholz, T., A. Kupper, and M. Schiffers. Quality of Context: What it is and why we need it. in 10th Workshop of the HP OpenView University Association (HPOVUA03). 2003. Geneva, Switzerland.
10. Sheikh, K., M. Wegdam, and M.v. Sinderen. Middleware Support for Quality of Context in Pervasive Context-Aware Systems. in IEEE International Workshop on Middleware Support for Pervasive Computing (PerWare'07). 2007. New York, USA.
11. Broens, T., D. Quartel, and M.v. Sinderen. Towards a Context Binding Transparency. in 13th EUNICE Open European Summer School. 2007. Enschede, the Netherlands: Springer.
12. Broens, T., et al., Dynamic Context Bindings in Pervasive Middleware, in Middleware Support for Pervasive Computing Workshop (PerWare'07). 2007: White Plains, USA.
13. Broens, T., A. Halteren, and M.v. Sinderen. Infrastructural Support for Dynamic Context Bindings. in 1st European Conference on Smart Sensing and Context (EuroSSc'06). 2006. Enschede, the Netherlands.
14. Broens, T. and A. van Halteren. SimuContext: simulating context sources for context-aware applications. in Intl. Conference on Networking and Services (ICNS06). 2006. Silicon Valley, USA.
15. Rubin, A.D. and D.E. Geer, Jr., Mobile Code Security. IEEE Internet Computing, 1998. 2(6): p. 30-34.
16. Blackstock, M., R. Lea, and C. Krasic. Towards Wide Area Interaction with Ubiquitous Computing Environments. in 1st European Conference on Smart Sensing and Context (EuroSSc'06). 2006. Enschede, the Netherlands.